

# Biometrics in Software Engineering

the what, the why, and the how

**Venera Arnaoudova**, Ph.D. | Pronouns: she/her/hers

Associate Professor

School of Electrical Engineering and Computer Science

Washington State University (WSU)

[Venera.Arnaoudova@wsu.edu](mailto:Venera.Arnaoudova@wsu.edu)

<http://www.veneraarnaoudova.com/>



WASHINGTON STATE  
UNIVERSITY

# The Software Engineering Lab and collaborators @ WSU



Venera Arnaoudova



Sarah Fakhoury



Ziyi Zhang



Devjeet Roy



Yuzhan Ma



Olusola Adesope

# Biometrics

Measurement and analysis of physiological states

## Examples

heart rate, brain activity (fNIRS, EEG, fMRI), eye movements (eyetracking), galvanic skin response (EDA)

## Widely used across multiple domains

E.g., medicine, psychology, education etc.



## Why biometrics for SE?

Biometrics give us insight into psychological and emotional states a person experiences during SE tasks

- Without biometrics, we must rely on self-reported measures which can be subject to bias

They can also be used as an interface to control systems (e.g., BCI)



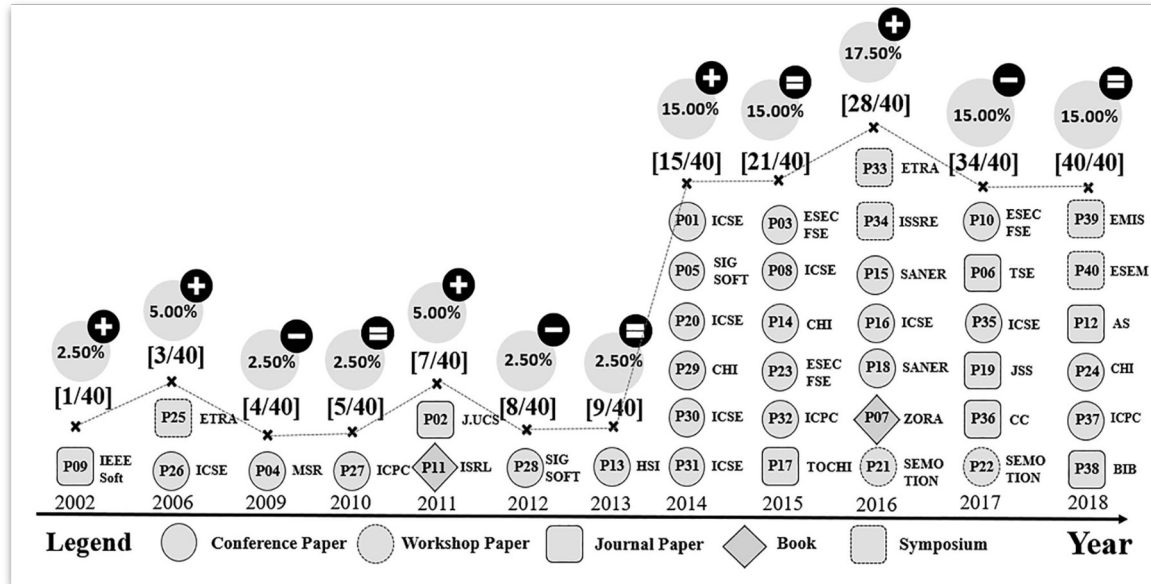
# Biometrics are quickly gaining adoption in SE research

Development of newer technologies has lead to:

- lowered the barrier to entry
- more precise measurements

Most commonly used biometrics:

- Eyetracking
- Brain Imaging - EEG, fMRI, fNIRS



Menzen, Juliano Paulo, Kleinner Farias, and Vinicius Bischoff. "Using biometric data in software engineering: a systematic mapping study." *Behaviour & Information Technology* 40.9 (2021): 880-902.

# Outline

1. Biometrics in SE
  - a. **Eyetracking**
  - b. Brain Imaging
    - i. fMRI
    - ii. **fNIRS**
    - iii. EEG
2. A novel framework to combine brain imaging with eyetracking
  - a. Overview
  - b. Tooling to facilitate multimodal biometric research in SE
3. Hands on!



# **Eyetracking**

# Eyetracking

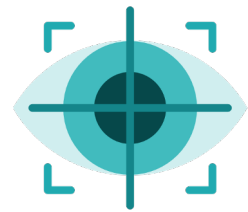


- Extensively used to study complex information processing tasks (such as SE)
- Low cost, and easy to deploy
- Eye movements give us insights into a participant's cognitive and psychological (including emotional) state
- **Next 10-15 years:** technologies that allow the usage of commodity hardware (e.g. smartphones, webcams) to capture eye movements





# Eyetracking: Basics



Types of eye movements:

- **Fixations:** Extended focus on a specific point of interest. This is where visual information processing occurs.
- **Saccades:** Rapid movement of the eyes from one point of interest to another. No processing is happening here.

Eyetrackers typically record the raw location at which the eyes are directed at some polling rate, known as **gazes**.

**Raw gazes** need to be processed into **fixations** and **saccades** using existing algorithms.

# Measures in Eyetracking



Wide variety of metrics that indicate different cognitive states

**Examples:**

fixation duration, fixation count, pupil diameter

## Eye-tracking Metrics in Software Engineering

Zohreh Sharafi  
Génie Informatique et Génie Logiciel  
École Polytechnique de Montréal  
Montréal, Canada  
zohreh.sharafi@polymtl.ca

Timothy Shaffer, Bonita Sharif  
Computer Science & Information Systems  
Youngstown State University  
Ohio, USA  
trshaffer@student.yzu.edu, bsharif@ysu.edu

Yann-Gaël Guéhéneuc  
Génie Informatique et Génie Logiciel  
École Polytechnique de Montréal  
Montréal, Canada  
yann-gael.gueheneuc@polymtl.ca

*Abstract*—Eye-tracking studies are getting more prevalent in software engineering. Researchers often use different metrics when publishing their results in eye-tracking studies. Even when the same metrics are used, they are given different names, causing difficulties in comparing studies. To encourage replications and facilitate advancing the state of the art, it is important that the metrics used by researchers be clearly and consistently defined in the literature. There is therefore a need for a survey of eye-tracking metrics to support the (future) goal of standardizing eye-tracking metrics. This paper seeks to bring awareness to the use of different metrics along with practical suggestions on using them. It compares and contrasts various eye-tracking metrics used in software engineering. It also provides definitions for common metrics and discusses some metrics that the software engineering community might borrow from other fields.

### I. INTRODUCTION

Researchers in software engineering (SE) use eye-tracking technology to study the cognitive processes and efforts involved in different types of SE tasks. An eye tracker (hardware and software) monitors an participant's visual attention via eye-movement data [1], [2]. Eye movements are essential to cognitive processes because they focus the participant's visual attention to the parts of a visual stimulus that are processed by the brain. Visual attention triggers cognitive processes that are required to perform tasks [3]. It is also a proxy for visual effort—a subset of cognitive effort—measured as the amount of visual attention allocated to parts of a visual stimulus. The

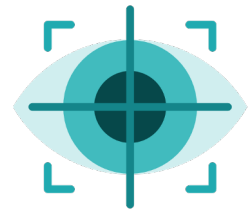
of fixations for the whole stimulus is called "Fixation Rate" [5], ON-target ALL-target [6], Ratio of ON-target:All-target Fixation (ROAF) [7], Ratio of fixation count [8], Relevant fixation count [9], and "Time in Region (TIR)" [10].

Consequently, we study exhaustively (to the best of our knowledge) all the ways in which an participant's visual effort has been measured in SE eye-tracking studies and provide unique names and definitions for the used metrics. We also discuss the interpretations of the values of these metrics with references to the literature. We provide practical suggestions on using these metrics and, finally, introduce a list of metrics that SE researchers could borrow from usability studies. Therefore, the contributions of this paper are:

- 1) Side-by-side comparisons and contrast of existing metrics for visual effort in SE eye-tracking studies.
- 2) A proposal of new metrics to borrow from other domains, with example applications.
- 3) A discussion on how to standardize metrics to help compare and replicate eye-tracking studies.

We provide necessary background information on eye tracking in Section II. Section III summarizes previous eye-tracking studies in SE. Section IV presents a list of visual-effort metrics followed by a discussion in Section V. Threats to the validity are reported in Section VI-A and VI-B. Section VII concludes and sketches future studies.

# Measures in Eyetracking



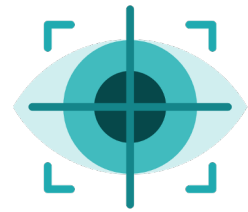
## Global metrics

Concerned with **areas of interest (AOI)** and associated eye movements

### Examples:

- Fixation duration/count - how many times did the participant view a particular area of interest (identifiers)
- Regression Rates - how often did the participant come back to a part of the stimulus they've already seen

# Measures in Eyetracking



## Scanpath metrics

Concerned with the sequence of eye movements

### Examples:

- Transition matrix - how often did the participant move between different areas of interest.
- Reading Order - how close are the participant's eye movements to a particular model for reading order (e.g. story/linear order vs execution order)


# iTrace: Eyetracking for SE



A set of tools for supporting eyetracking experiments in SE

Allows mapping of eye gazes to fine-grained source code elements, and common gaze processing algorithms

Support for Eclipse, Visual Studio, Atom, and other IDEs/editors



## iTrace: Enabling Eye Tracking on Software Artifacts within the IDE to Support Software Engineering Tasks

Timothy R. Shaffer<sup>†</sup>, Jenna L. Wise<sup>†</sup>, Braden M. Walters<sup>†</sup>, Sebastian C. Müller<sup>‡</sup>, Michael Falcone<sup>†</sup>, Bonita Sharif<sup>†</sup>

<sup>†</sup>Youngstown State University, USA  
Department of CS and IS  
{trshaffer,jlwise,bmwalters01}@student.yosu.edu  
mrfalcone@student.yosu.edu, bsharif@ysu.edu

<sup>‡</sup>University of Zurich, Switzerland  
Department of Informatics  
smueller@ifi.uzh.ch

### ABSTRACT

The paper presents *iTrace*, an Eclipse plugin that implicitly records developers' eye movements while they work on change tasks. *iTrace* is the first eye tracking environment that makes it possible for researchers to conduct eye tracking studies on large software systems. An overview of the design and architecture is presented along with features and usage scenarios. *iTrace* is designed to support a variety of eye trackers. The design is flexible enough to record eye movements on various types of software artifacts (Java code, text/html/xml documents, diagrams), as well as IDE user interface elements. The plugin has been successfully used for software traceability tasks and program comprehension tasks. *iTrace* is also applicable to other tasks such as code summarization and code recommendations based on developer eye movements. A short video demonstration is available at <https://youtu.be/30UnLCK4dXo>.

### Categories and Subject Descriptors

D.2.7 [Software Engineering]: Distribution, Maintenance, and Enhancement

### General Terms

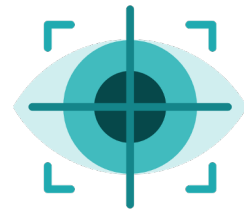
Experimentation, Measurement, Human Factors

researcher to collect eye movements of software engineers while they work on software tasks such as adding a new feature, fixing a bug, or on a general comprehension task. The eye movement data is used to study the cognitive thought processes [6] of developers as they perform a task using different software artifacts. Existing eye-tracking studies have mainly studied developers comprehending software artifacts such as source code, models such as UML diagrams, and software visualizations.

An eye tracker consists of both hardware and software. The hardware is a physical device that usually sits under the monitor. The software provided by eye tracking vendors is in the form of an experiment builder (e.g., Tobii Studio from Tobii Inc.) that allows researchers to build the experiment workflow using various stimuli such as a still image, a website, a video recording, or free form desktop recording.

Most existing eye tracking studies (besides [11] and [4]) done in the SE community use small snippets of code shown as an image to study participants. An ad hoc system to support scrolling using slider bar events for a few small programs was done in [9] but the tool is unavailable. In all other studies, the image needs to be displayed on the screen all at once and participants are not allowed to scroll as scrolling would interfere with data collected and make post processing extremely difficult if not impossible. This is because the eye tracker is not aware of the type of stimulus presented to it. It reports the (x,y) coordinates where a person is look

# Notable Publications



No difference between underscore and camel case in terms of performance.

## Metrics Used

Fixation duration, fixation count, AOI transitions

Sharif, Bonita, and Jonathan I. Maletic. "An eye tracking study on camelcase and under\_score identifier styles." *2010 IEEE 18th International Conference on Program Comprehension*. IEEE, 2010.

18th IEEE International Conference on Program Comprehension

## An Eye Tracking Study on camelCase and under\_score Identifier Styles

Bonita Sharif and Jonathan I. Maletic  
Department of Computer Science  
Kent State University  
Kent, Ohio 44242

bimoes@cs.kent.edu and jmaletic@cs.kent.edu

**Abstract**— An empirical study to determine if identifier-naming conventions (i.e., camelCase and under\_score) affect code comprehension is presented. An eye tracker is used to capture quantitative data from human subjects during an experiment. The intent of this study is to replicate a previous study published at ICPC 2009 (Binkley et al.) that used a timed response test method to acquire data. The use of eye-tracking equipment gives additional insight and overcomes some limitations of traditional data gathering techniques. Similarities and differences between the two studies are discussed. One main difference is that subjects were trained mainly in the underscore style and were all programmers. While results indicate no difference in accuracy between the two styles, subjects recognize identifiers in the underscore style more quickly.

**Keywords**— identifier styles; eye-tracking study; code readability

### I. INTRODUCTION

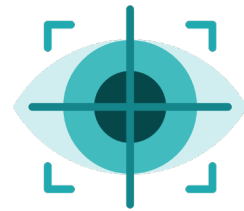
The comprehension of identifier names in programs is at the core of program understanding. Identifier names are

identifier. The position of the underscore on the keyboard and the number and combination of keystrokes required plays a role in typing speed. However, does the ease of writing identifiers affect the accuracy of code readability and maintainability?

To address this topic, Binkley et al. [4] conducted a study with 135 subjects consisting of programmers and non-programmers to determine which identifier style was faster and more accurate. They hypothesized that identifier style affects the speed and accuracy of software maintenance. The subjects (who had programming experience) were mostly trained in the camel-case style. The study used an online game-like interface to gather timed responses from the subjects. Their findings show that camel-cased identifiers lead to higher accuracy among all subjects, and those trained in the camel-case style, were able to recognize camel-cased identifiers faster. However, with respect to all subjects, camel-cased identifiers took 13.5% longer than underscored identifiers ( $p\text{-value} < 0.0001$ ).

Here, we attempt to replicate Binkley et al.'s [4] experiment using an eye tracker to gather eye gaze data during the experiment. In our study, only programmers

# Notable Publications



Proposes a new metric *linearity*, which is shown to have a strong effect on the reading order

## Metrics Used

Fixation duration, fixation count, regression rates, story/execution order

Peitek, Norman, Janet Siegmund, and Sven Apel. "What drives the reading order of programmers? an eye tracking study." *Proceedings of the 28th International Conference on Program Comprehension*. 2020.

2020 IEEE/ACM 28th International Conference on Program Comprehension (ICPC)

## What Drives the Reading Order of Programmers? An Eye Tracking Study

Norman Peitek  
Leibniz Institute for Neurobiology  
Magdeburg, Germany

Janet Siegmund  
Chemnitz University of Technology  
Chemnitz, Germany

Sven Apel  
Saarland University  
Saarbrücken, Germany

### ABSTRACT

**Background:** The way how programmers comprehend source code depends on several factors, including the source code itself and the programmer. Recent studies showed that novice programmers tend to read source code more like natural language text, whereas experts tend to follow the program execution flow. But, it is unknown how the *linearity of source code* and the comprehension strategy influence programmers' *linearity of reading order*.

**Objective:** We replicate two previous studies with the aim of additionally providing empirical evidence on the influencing effects of linearity of source code and programmers' comprehension strategy on linearity of reading order.

**Methods:** To understand the effects of linearity of source code on reading order, we conducted a non-exact replication of studies by Busjahn et al. and Peachock et al., which compared the reading order of novice and expert programmers. Like the original studies, we used an eye-tracker to record the eye movements of participants (12 novice and 19 intermediate programmers).

**Results:** In line with Busjahn et al. (but different from Peachock et al.), we found that experience modulates the reading behavior of participants. However, the linearity of source code has an even stronger effect on reading order than experience, whereas the com-

### ACM Reference Format:

Norman Peitek, Janet Siegmund, and Sven Apel. 2020. What Drives the Reading Order of Programmers? An Eye Tracking Study. In *28th International Conference on Program Comprehension (ICPC '20)*, October 5–6, 2020, Seoul, Republic of Korea. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3387904.3389279>

### 1 INTRODUCTION

In the past decades, much research has focused on how programmers comprehend source code, which is a central activity in software development [26, 52]. The underlying cognitive process, *program comprehension*, is a prerequisite for all subsequent programmer activities, such as testing, debugging, and maintenance. Past research theorized on two main strategies of how programmers comprehend software. *Bottom-up comprehension* is used when programmers lack domain knowledge, experience, or context to efficiently understand source code [39]. Instead, they have to understand individual source code lines and statements and integrate their semantic meaning to eventually build an overarching understanding (i.e., chunking [46]). *Top-down comprehension* is used when programmers take advantage of previous experience or do-



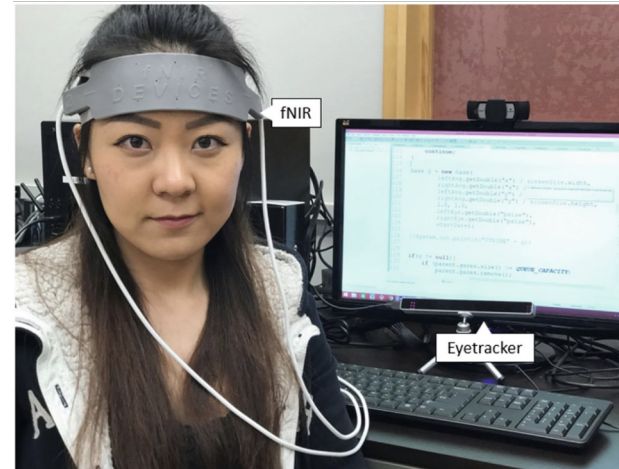
# Brain Imaging



# Brain Imaging



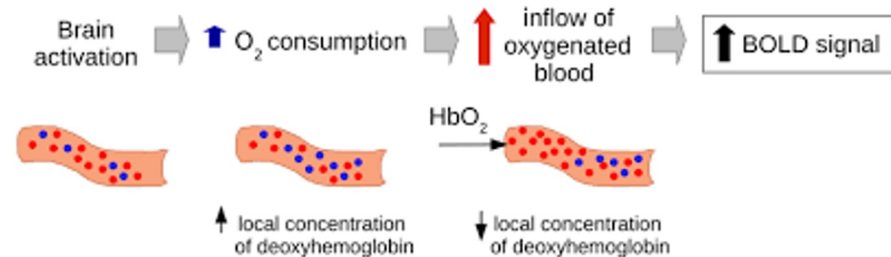
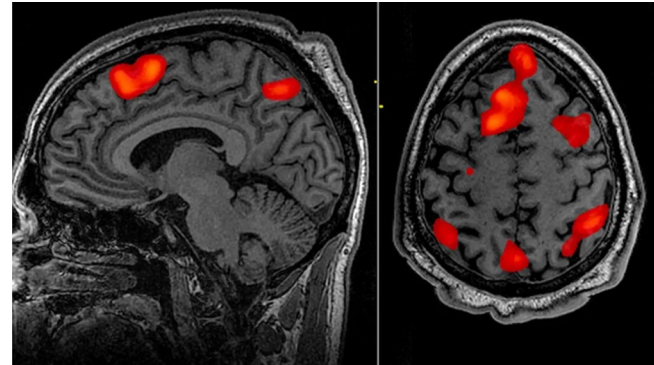
- Direct measurements on brain activity
- Higher accuracy and precision than eye-tracking and other biometrics for cognitive effort
- Allows us to localize brain activity to different brain regions and study interconnectivity between these regions
- Requires more expertise to operate (difficult to use as blackbox tool)



# Functional Magnetic Resonance Imaging (fMRI)



- Activated brain regions need more oxygen
- By measuring oxygenated and deoxygenated blood we can approximate **mental effort**
- Uses strong magnets to create structural images of the human brain



# Notable Publications



## Understanding Understanding Source Code with Functional Magnetic Resonance Imaging

Janet Siegmund<sup>\*,\*</sup>, Christian Kästner<sup>\*,</sup>, Sven Apel<sup>\*,</sup>, Chris Parnin<sup>\*,</sup>, Anja Bethmann<sup>\*,</sup>,  
Thomas Leich<sup>\*,</sup>, Gunter Saake<sup>\*,</sup>, and André Brechmann<sup>\*,</sup>  
<sup>\*</sup>University of Passau, Germany   <sup>†</sup>Carnegie Mellon University, USA  
<sup>‡</sup>Georgia Institute of Technology, USA   <sup>§</sup>Leibniz Inst. for Neurobiology Magdeburg, Germany  
<sup>¶</sup>Metop Research Institute, Magdeburg, Germany   <sup>||</sup>University of Magdeburg, Germany

### ABSTRACT

Program comprehension is an important cognitive process that inherently eludes direct measurement. Thus, researchers are struggling with providing suitable programming languages, tools, or coding conventions to support developers in their everyday work. In this paper, we explore whether *functional magnetic resonance imaging (fMRI)*, which is well established in cognitive neuroscience, is feasible to more directly measure program comprehension. In a controlled experiment, we observed 17 participants inside an fMRI scanner while they were comprehending short source-code snippets, which we contrasted with locating syntax errors. We found a clear, distinct activation pattern of five brain regions, which are related to working memory, attention, and language processing—all processes that fit well to our understanding of program comprehension. Our results encourage us and, hopefully, other researchers to use fMRI in future studies to measure program comprehension and, in the long run, answer questions, such as: Can we predict whether someone will be an excellent programmer? How effective are new languages and tools for program understanding? How should we train developers?

### 1. INTRODUCTION

As the world becomes increasingly dependent on the billions lines of code written by software developers, little comfort can be taken in the fact that we still have no fundamental understanding of

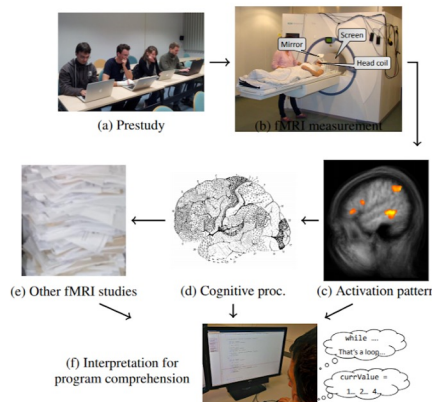


Figure 1: Workflow of our fMRI study.

First study using fMRI in SE

- Reading source code activates brain areas related to **language comprehension, problem solving, working memory.**

Siegmund, Janet, et al. "Understanding understanding source code with functional magnetic resonance imaging." *Proceedings of the 36th international conference on software engineering*. 2014.

# Notable Publications



- Replicates prior work and supports the use of fMRI for program comprehension studies
- Comprehension based on semantic cues elicits lower levels of brain activation compared to bottom-up comprehension

Siegmund, Janet, et al. "Measuring neural efficiency of program comprehension." *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 2017.



## Measuring Neural Efficiency of Program Comprehension

Janet Siegmund  
University of Passau  
Passau, Germany

Norman Peitek  
Leibniz Institute for Neurobiology  
Magdeburg, Germany

Chris Parnin  
NC State University  
Raleigh, North Carolina, USA

Sven Apel  
University of Passau  
Passau, Germany

Johannes Hofmeister  
University of Passau  
Passau, Germany

Christian Kästner  
Carnegie Mellon University  
Pittsburgh, Pennsylvania, USA

Andrew Begel  
Microsoft Research  
Redmond, Washington, USA

Anja Bethmann  
Leibniz Institute for Neurobiology  
Magdeburg, Germany

André Brechmann  
Leibniz Institute for Neurobiology  
Magdeburg, Germany

### ABSTRACT

Most modern software programs cannot be understood in their entirety by a single programmer. Instead, programmers must rely on a set of cognitive processes that aid in seeking, filtering, and shaping relevant information for a given programming task. Several theories have been proposed to explain these processes, such as "beacons," for locating relevant code, and "plans," for encoding cognitive models. However, these theories are decades old and lack validation with modern cognitive-neuroscience methods. In this paper, we report on a study using functional magnetic resonance imaging (fMRI) with 11 participants who performed program comprehension tasks. We manipulated experimental conditions related to beacons and layout to isolate specific cognitive processes related to bottom-up comprehension and comprehension based on semantic cues. We found evidence of semantic chunking during bottom-up comprehension and lower activation of brain areas during comprehension based on semantic cues, confirming that beacons ease comprehension.

### CCS CONCEPTS

• Human-centered computing → HCI design and evaluation methods; Empirical studies in HCI;

### KEYWORDS

functional magnetic resonance imaging; program comprehension

*Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Paderborn, Germany, September 4–8, 2017 (ESEC/FSE'17), 11 pages.*  
<https://doi.org/10.1145/3106237.3106268>

### 1 INTRODUCTION

During program comprehension, the eyes of programmers glide across a computer screen. In just seconds, they can extract a deep understanding from abstract symbols and text arranged in a source-code file. Expert programmers are especially adept at program comprehension of familiar code—their eyes dance around, finding points of interest, called *beacons* (or semantic cues) that provide hints about a program's purpose, such as method signatures, and common programming idioms. *Top-down comprehension* has been used as an umbrella term to describe cognitive processes related to experience and expectation that guide the understanding of source code [7]. Researchers have also theorized that programmers must use preformed knowledge structures, called *plans*, that represent semantic and syntactical patterns of software [8]. For example, an identifier `bubbleSort` indicates the presence of a sorting algorithm and primes a programmer to expect other elements of the bubble-sort algorithm, such as code related to a swap of array elements.

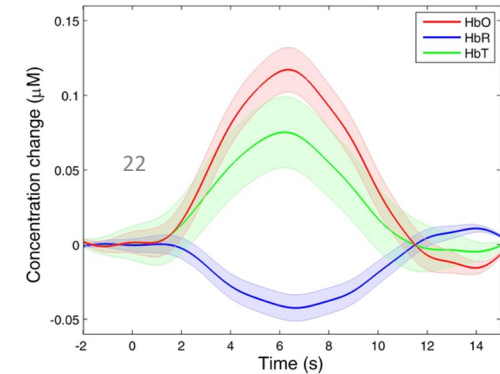
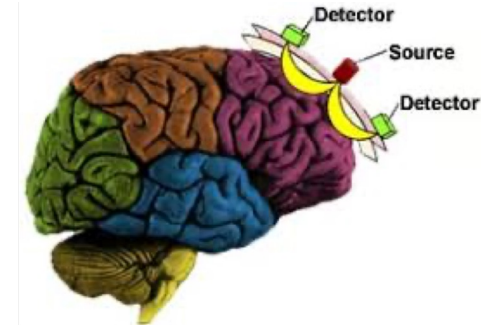
In contrast, when code lacks familiar semantic cues, program-



# Functional Near-Infrared Spectroscopy (fNIRS)



- fNIRS measures same hemodynamic response as fMRI
  - Concentration changes in oxygenated (HbO) and deoxygenated hemoglobin (HbR) indicators of mental effort
- Used in a wide variety of working memory research
- Less invasive
- Allows us to more closely replicate real working environments + tasks!

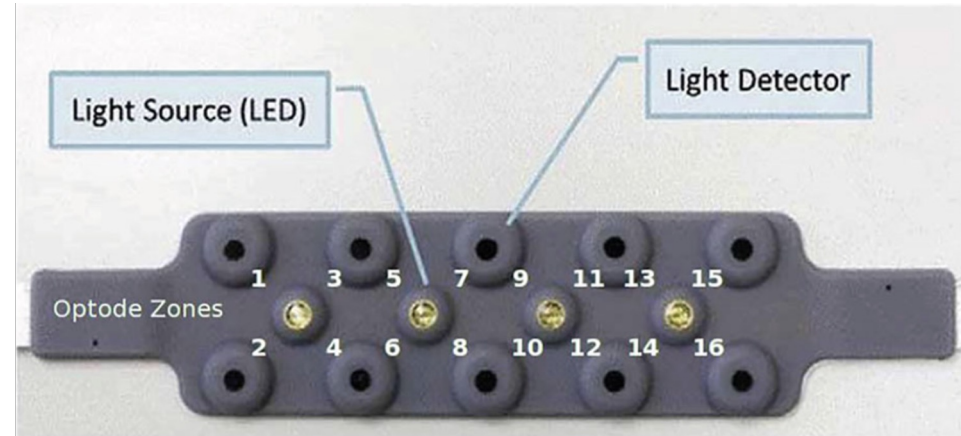




# fNIRS: Basics



- Infrared light is emitted by several sensors, and after going through tissue, some of this gets reabsorbed by detectors
- The headband is divided into multiple **optodes**



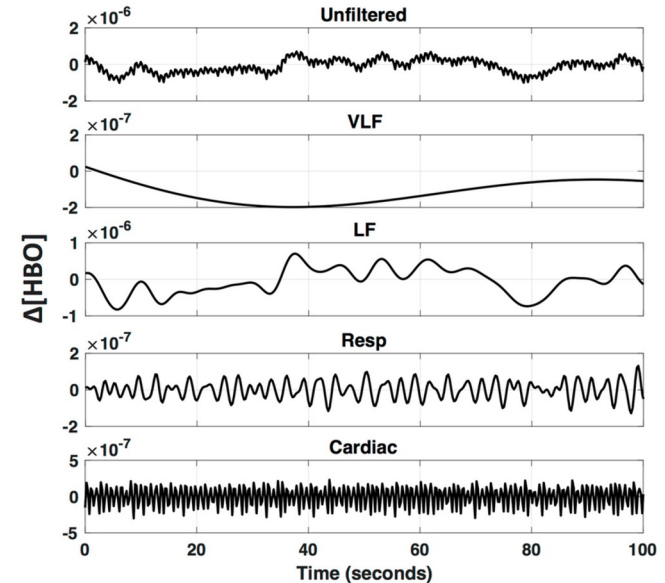
Biopac fNIRS 2000s



# Processing fNIRS Data



- Signal Quality (SNR)
  - Channel exclusions (poorly fit optodes)
    - Visual Inspection, PHEOBE
  - Motion artifact correction
    - Visual inspection + Wavelet filtering (high freq. spikes)
- Physiological Noise
  - Filtering, denoising, baseline drift correction
  - Bandpass filtering ( $> 0.2$  and  $< 0.01$ Hz heartbeat, respiratory resp)
  - LF de-noising (SSD, PCA, GloAvg)
- Detecting task evoked activation
  - Using both Hbo, HbR and combination (Oxy)
  - Block Averaging control vs task (t-test, ANOVA)
  - GLM
- Community continues to work towards standardized methodologies



Hocke, Lia M., et al. "Automated processing of fNIRS data—a visual guide to the pitfalls and consequences." *Algorithms* 11.5 (2018): 67.  
Klein, Franziska, and Cornelia Kranczioch. "Signal processing in fNIRS: a case for the removal of systemic activity for single trial data." *Frontiers in human neuroscience* 13 (2019): 331.

# Electroencephalography (EEG)



- Measures electrical impulses in the brain
- Different patterns of oscillations of these impulses or *brain waves* have been shown to be associated with distinct cognitive states
- Measures neuronal activity directly, but signal can be quite noisy



# Notable Publications



## Using Psycho-Physiological Measures to Assess Task Difficulty in Software Development

Thomas Fritz<sup>†</sup>, Andrew Begel<sup>\*</sup>, Sebastian C. Müller<sup>†</sup>, Serap Yigit-Elliott<sup>\*</sup>, Manuela Züger<sup>†</sup>

<sup>†</sup>University of Zurich  
Zurich, Switzerland

<sup>\*</sup>Microsoft Research  
Redmond, WA USA

<sup>\*</sup>Exponent  
Bellevue, WA USA

### ABSTRACT

Software developers make programming mistakes that cause serious bugs for their customers. Existing work to detect problematic software focuses mainly on *post hoc* identification of correlations between bug fixes and code. We propose a new approach to address this problem — detect when software developers are experiencing difficulty while they work on their programming tasks, and stop them before they can introduce bugs into the code.

In this paper, we investigate a novel approach to classify the difficulty of code comprehension tasks using data from psycho-physiological sensors. We present the results of a study we conducted with 15 professional programmers to see how well an eye-tracker, an electrodermal activity sensor, and an electroencephalography sensor could be used to predict whether developers would find a task to be difficult.

### 1. INTRODUCTION

Knowing how hard a task is as it is being performed can help in many dimensions. For instance, the estimate for completing a task might be revised or the likelihood of a bug occurring in the source code changes for the task might be predicted. Existing work to determine task difficulty has mainly focused on already existing artifacts, such as task descriptions, and the similarity of artifacts using machine learning classifiers. In our research, we are investigating a novel approach to determine task difficulty that uses psycho-physiological data gathered from the developer while he is working, such as electroencephalographic (EEG) activity along the forehead or electrodermal activity (EDA). By using psycho-physiological sensors and collecting data while a developer is performing a task, we present the first approach that can support an instantaneous measure of task

- Among the first studies to combine multiple biometrics – EEG, EDA, eyetracking
- EEG and eye tracking metrics can distinguish between task difficulty at coarse level

Fritz, Thomas, et al. "Using psycho-physiological measures to assess task difficulty in software development." *Proceedings of the 36th international conference on software engineering*. 2014.

# Notable Publications



- Experienced programmers comprehend programs with lower cognitive load than less experienced programmers
- Self reported programming efficiency better correlated with efficiency of comprehension when compared to years of experience
- EEG and eyetracking



## Correlates of Programmer Efficacy and Their Link to Experience: A Combined EEG and Eye-Tracking Study

Norman Peitek  
Saarland University,  
Saarland Informatics Campus  
Saarbrücken, Germany

Annabelle Bergum  
Saarland University,  
Saarland Informatics Campus,  
Graduate School of Computer Science  
Saarbrücken, Germany

Maurice Rekrut  
German Research Center for Artificial  
Intelligence, Saarland Informatics  
Campus, Saarbrücken, Germany

Jonas Mucke  
Chemnitz University of Technology  
Chemnitz, Germany

Matthias Nadig  
German Research Center for Artificial  
Intelligence, Saarland Informatics  
Campus, Saarbrücken, Germany

Chris Parnin  
NC State University  
Raleigh, North Carolina, USA

Janet Siegmund  
Chemnitz University of Technology  
Chemnitz, Germany

Sven Apel  
Saarland University,  
Saarland Informatics Campus  
Saarbrücken, Germany

### ABSTRACT

**Background:** Despite similar education and background, programmers can exhibit vast differences in efficacy. While research has identified some potential factors, such as programming experience and domain knowledge, the effect of these factors on programmers' efficacy is not well understood.

**Aims:** We aim at unraveling the relationship between efficacy (speed and correctness) and measures of programming experience. We further investigate the correlates of programmer efficacy in terms of reading behavior and cognitive load.

**Method:** For this purpose, we conducted a controlled experiment

### KEYWORDS

Programmer efficacy, program comprehension, cognitive load, electroencephalography, eye tracking

### ACM Reference Format:

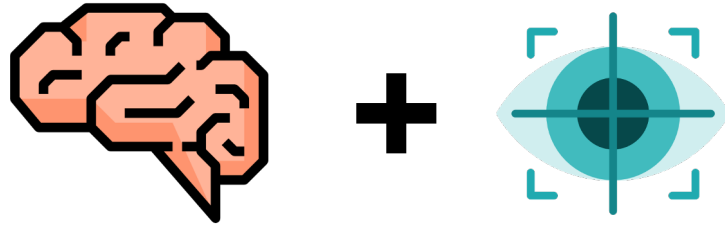
Norman Peitek, Annabelle Bergum, Maurice Rekrut, Jonas Mucke, Matthias Nadig, Chris Parnin, Janet Siegmund, and Sven Apel. 2022. Correlates of Programmer Efficacy and Their Link to Experience: A Combined EEG and Eye-Tracking Study. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22)*, November 14–18, 2022, Singapore, Singapore. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3540250.3549084>

# Comparison of brain imaging techniques



	fMRI	fNIRS	EEG
Measures	BOLD	BOLD	Electrical activity
Delay	Several seconds	Several seconds	Milliseconds
Temporal Resolution	~1–2 seconds	~1–2 seconds	Milliseconds
Spatial Resolution	++	+	—
Participant Restrictions	—	+	++
Environmental Limitations	—	++	+
Portable	No	Yes*	Yes*
Financial Costs	—	+	+

# Combining Brain Imaging with Eyetracking

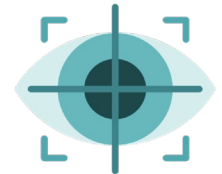


# Eye tracking to improve explanatory power of Neural Imaging

- A single measurement method is not sufficient to understand the full complexity of the cognitive processes involved in program comprehension
- Using a multi-modal approach to connect the cognitive processes to behavior. i.e. connect eye movement patterns and brain activation to the programmer's strategy of program comprehension

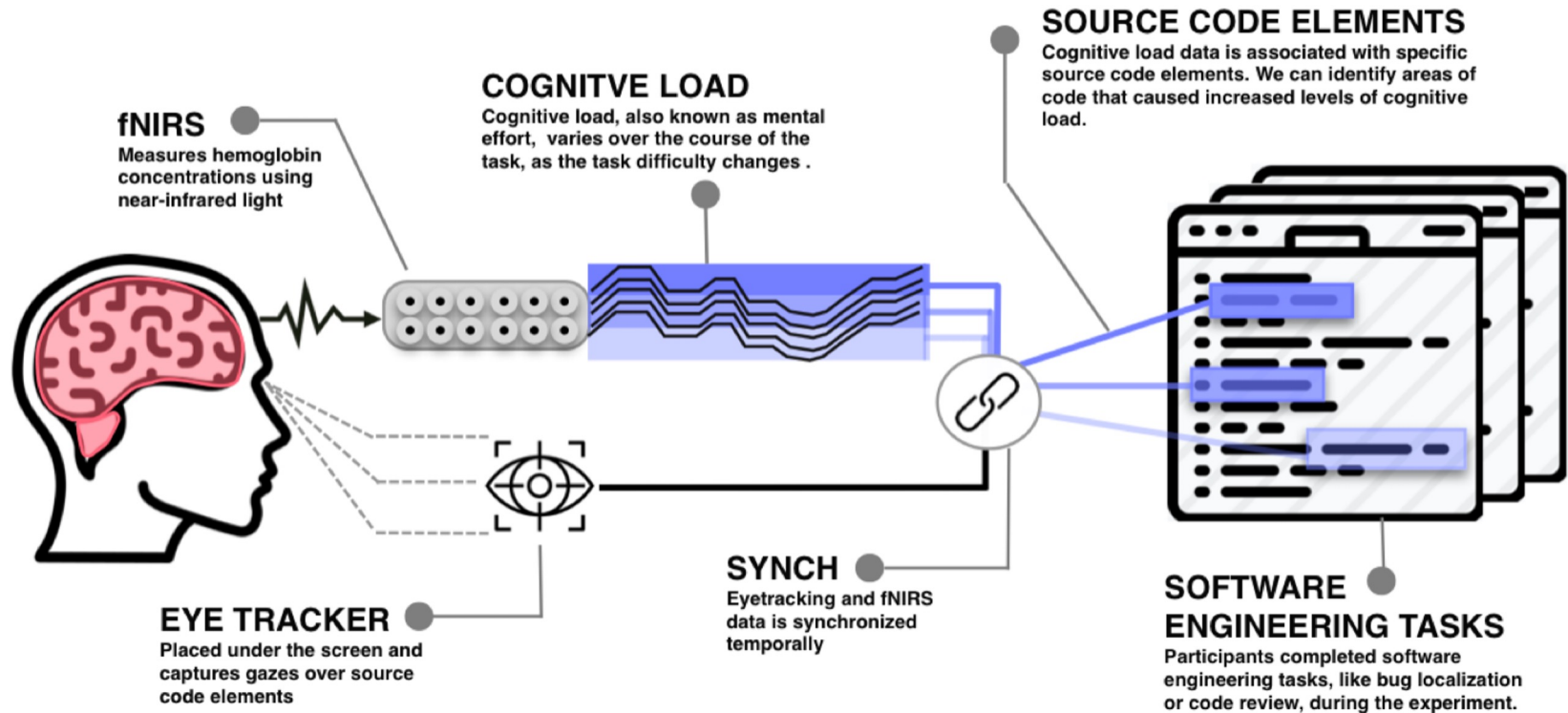


**Brain Imaging**  
*Cognitive Load*



**Eye Tracking**  
*Specific identifiers  
in the Source code*

# A novel framework for program comprehension research





# A novel framework for program comprehension research

ICPC '18

## The Effect of Poor Source Code Lexicon and Readability on Developers' Cognitive Load

Sarah Fakhoury, Yuzhan Ma  
Venera Arnaoudova  
Washington State University  
School of Electrical Engineering and Computer Science  
{sfakhour, yma1, varnaoud}@eecs.wsu.edu

Olusola Adesope  
Washington State University  
Educational Psychology Program, College of Education  
olusola.adesope@wsu.edu

### ABSTRACT

It has been well documented that a large portion of the cost of any software lies in the time spent by developers in understanding a program's source code before any changes can be undertaken. One of the main contributors to software comprehension, by subsequent developers or by the authors themselves, has to do with the quality of the lexicon, (i.e., the identifiers and comments) that is used by developers to embed domain concepts and to communicate with their teammates. In fact, previous research shows that there is a positive correlation between the quality of identifiers and the quality of a software project. Results suggest that poor quality lexicon impairs program comprehension and consequently increases the effort that developers must spend to maintain the software. However, we do not yet know or have any empirical evidence, of the relationship between the quality of the lexicon and the cognitive load that developers experience when trying to understand a piece of software. Given the associated costs, there is a critical need to empirically characterize the impact of the quality of the lexicon on developers' ability to comprehend a program.

### ACM Reference Format:

Sarah Fakhoury, Yuzhan Ma, Venera Arnaoudova, and Olusola Adesope. 2018. The Effect of Poor Source Code Lexicon and Readability on Developers' Cognitive Load. In *ICPC '18: 26th IEEE/ACM International Conference on Program Comprehension, May 27–28, 2018, Gothenburg, Sweden*. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3196321.3196347>

### 1 INTRODUCTION

Program comprehension is a fundamental activity within the software development life cycle. An important contributor to software comprehension has to do with the quality of the lexicon, i.e., the identifiers (names of programming entities such as classes or variables) and comments that are used by developers to embed domain concepts and to communicate with their teammates. Previous studies show that source code contains 42% of the domain terms [19] meaning that the lexicon is a way to express understanding of the problem domain and solution, and comment upon the ideas that

ACM SIGSOFT  
distinguished  
paper award

Fakhoury, Sarah, et al. "The effect of poor source code lexicon and readability on developers' cognitive load." *Proceedings of the 26th Conference on Program Comprehension*. 2018.

# Linguistic Antipatterns (LAs)

Poor recurring practices that create inconsistencies between **naming**, **documentation**, and **implementation** of the software

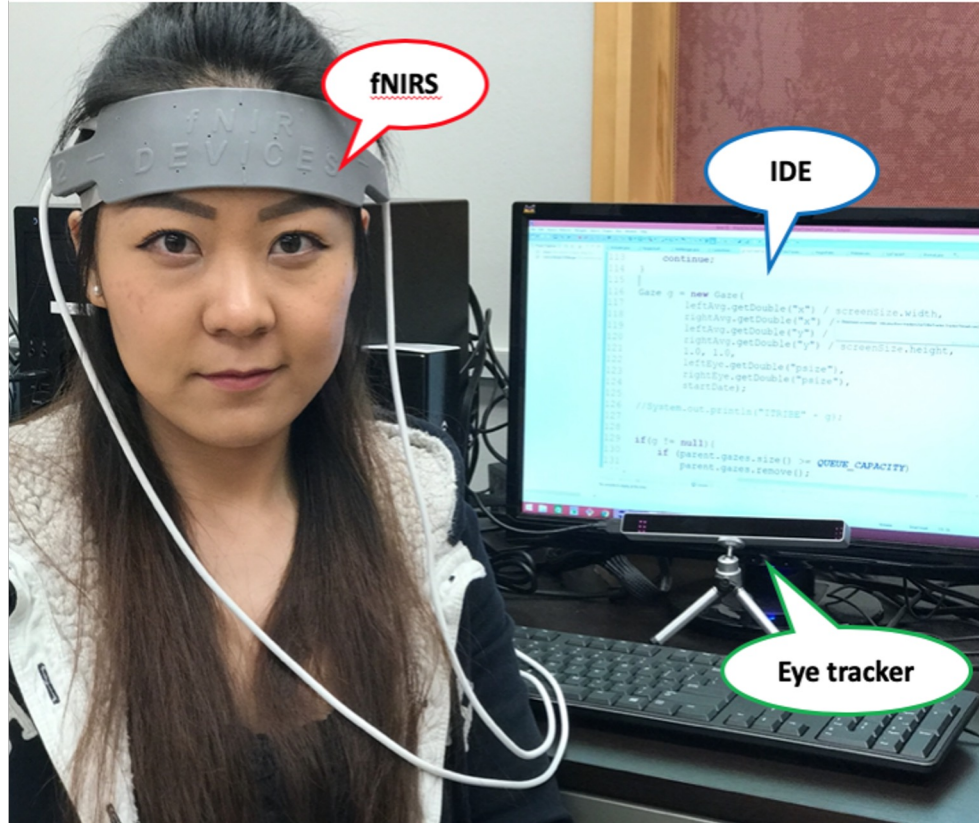
```
int[] isReached;
```

```
private static boolean _stats = true;
```

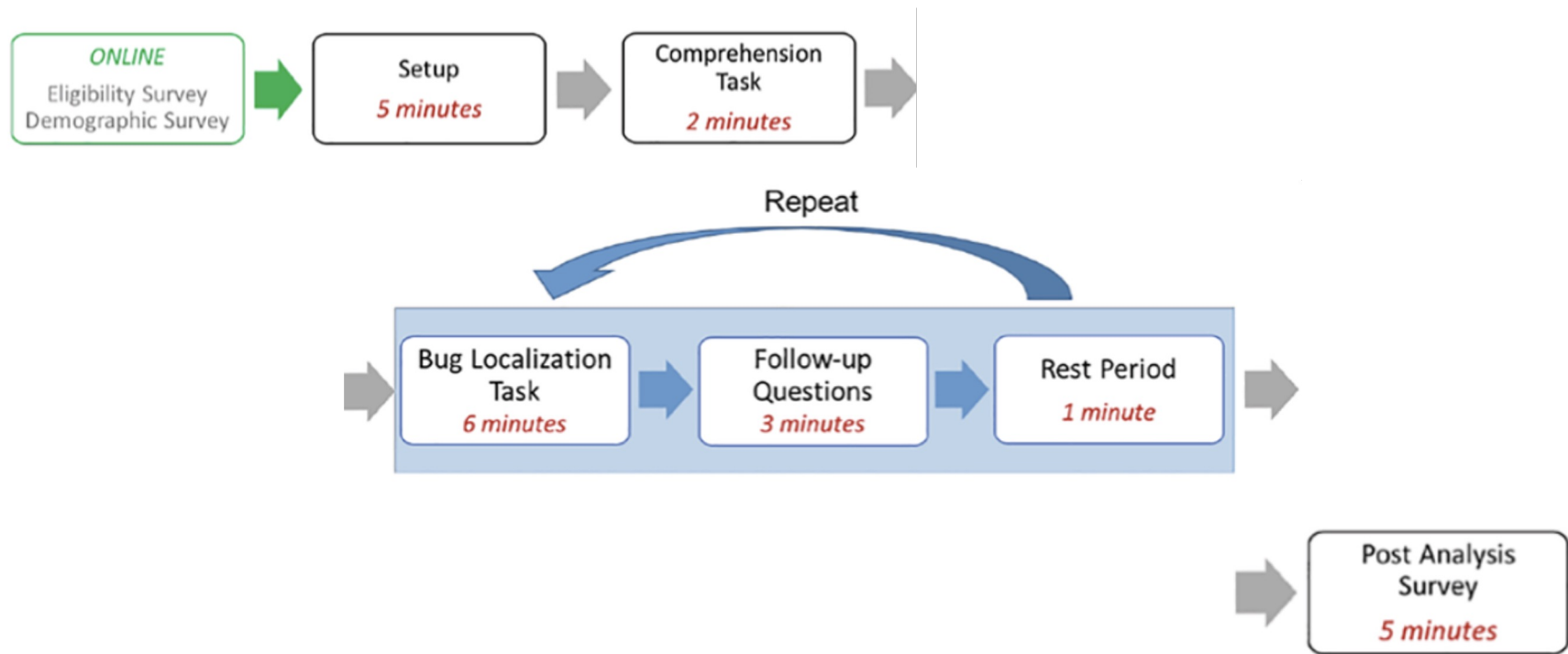
# Experiment Design

- Assign participants to a **control** group or **treatment** group
- Each participant performs a bug localization task on several code snippets
  - We collect open source code snippets and inject bugs into them
- Treatment group viewed same code as control group, but we inject **linguistic anti-patterns (LAs)** into the code
  - This isolates any increased neuronal activity in the treatment group to the LAs injected into the code

# Experiment Setup

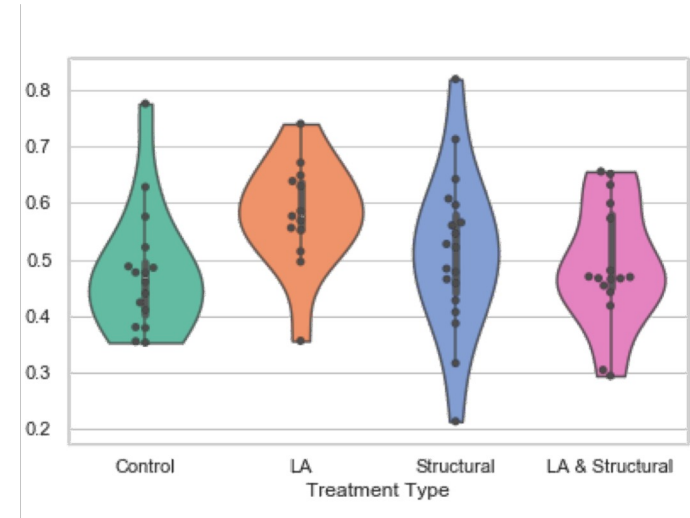


# Experiment Protocol



# Results: Impact of Linguistic Antipatterns

- **Correctness:** 65% of participants are successful at finding the bug. Compared to 77% of participants for the control task.
- **Speed:** Takes almost twice as long to find the same bug. 5 minutes vs 3 minutes for control
- **Cognitive Impact:** Poor lexicon significantly increased cognitive load, measured by HbO and HbR concentrations ( $p=0.005$  with large effect size)
- Not immediately obvious to participants why the code was causing inconsistencies in their mental model



# ...and more!

EMSE '20

## Measuring the impact of lexical and structural inconsistencies on developers' cognitive load during bug localization

Sarah Fakhoury<sup>1</sup> · Devjeet Roy<sup>1</sup> · Yuzhan Ma<sup>2</sup> · Venera Arnaudova<sup>1</sup> · Olusola Adesope<sup>3</sup>

Published online: 8 August 2019

© Springer Science+Business Media, LLC, part of Springer Nature 2019

### Abstract

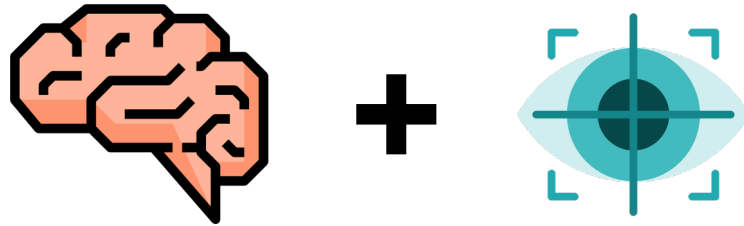
A large portion of the cost of any software lies in the time spent by developers in understanding a program's source code before any changes can be undertaken. Measuring program comprehension is not a trivial task. In fact, different studies use self-reported and various psycho-physiological measures as proxies. In this research, we propose a methodology using functional Near Infrared Spectroscopy (fNIRS) and eye tracking devices as an objective measure of program comprehension that allows researchers to conduct studies in environments close to real world settings, at identifier level of granularity. We validate our methodology and apply it to study the impact of lexical, structural, and readability issues on developers' cognitive load during bug localization tasks. Our study involves 25 undergraduate and graduate students and 21 metrics. Results show that the existence of lexical inconsistencies in the source code significantly increases the cognitive load experienced



- Both lexical and structural inconsistencies cause reduced performance, but only lexical inconsistencies result in increased cognitive load
- Self report task difficulty, total fixation duration and cognitive load are not aligned; they seem to measure different aspects of task difficulty

Fakhoury, Sarah, et al. "Measuring the impact of lexical and structural inconsistencies on developers' cognitive load during bug localization." *Empirical Software Engineering* 25 (2020): 2140-2178.

# Tool Support for Biometrics in SE

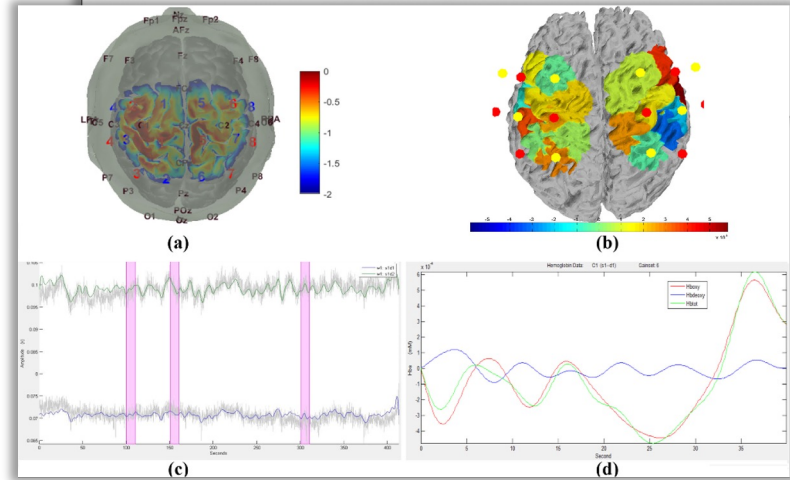
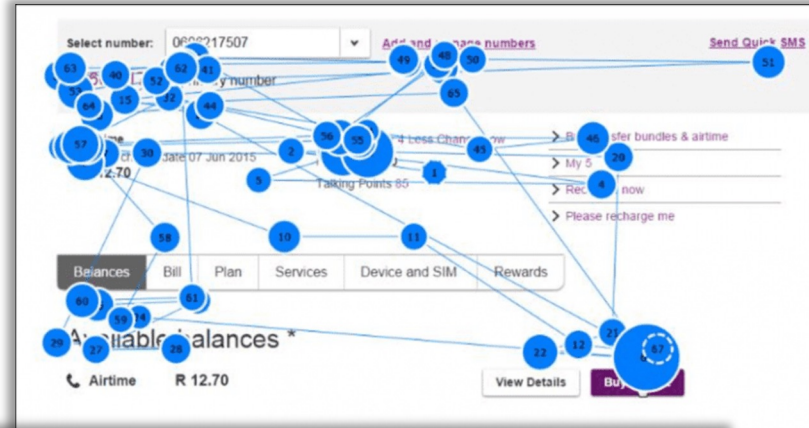




# Limitations of existing tools

- Existing fNIRS toolkits do not combine multimodal data like eye tracking
- Existing tools for eye tracking map gazes over images, not source code elements!

**Need new tools that allow dynamic visualization of multimodal data**



# VITALSE: Data Visualization and Analysis for Biometrics in SE

ICSE '20

## VITALSE: Visualizing Eye Tracking and Biometric Data

Devjeet Roy  
devjeet.roy@wsu.edu  
Washington State University  
Pullman, Washington

Sarah Fakhoury  
sarah.fakhoury@wsu.edu  
Washington State University  
Pullman, Washington

Venera Arnaoudova  
venera.arnaoudova@wsu.edu  
Washington State University  
Pullman, Washington

### ABSTRACT

Recent research in empirical software engineering is applying techniques from neurocognitive science and breaking new grounds in the ways that researchers can model and analyze the cognitive processes of developers as they interact with software artifacts. However, given the novelty of this line of research, only one tool exists to help researchers represent and analyze this kind of multi-modal biometric data. While this tool does help with visualizing temporal eyetracking and physiological data, it does not allow for the mapping of physiological data to source code elements, instead projecting information over images of code. One drawback of this is that researchers are still unable to meaningfully combine and map physiological and eye tracking data to source code artifacts. The use of images also bars the support of long or multiple code files, which prevents researchers from analyzing data from experiments conducted in realistic settings. To address these drawbacks, we propose VITALSE, a tool for the interactive visualization of combined multi-modal biometric data for software engineering tasks. VITALSE provides interactive and customizable temporal heatmaps created with synchronized eyetracking and biometric data. The tool supports analysis on multiple files, user defined annotations for points of interest over source code elements, and high level customizable metric summaries for the provided dataset. VITALSE,

electrodermal activity (EDA) to investigate task difficulty during code comprehension.

The use of neuroimaging in the domain is still in its infancy. In his keynote at the International Conference on Program Comprehension (ICPC'19), Westley Weimer summarizes the less than a dozen papers so far published using high-resolution medical imaging technologies and highlights the "game changing" areas in program comprehension that open up opportunities for new research [11]. The papers published thus far, use either functional Magnetic Resonance Imaging (fMRI) or functional Near Infrared Spectroscopy (fNIRS) techniques. In the last couple of years, researchers started combining eye tracking and brain imaging techniques together [3, 4, 7]. However, in this novel integration of brain imaging techniques in empirical software engineering studies, researchers rely on tools developed for neurocognitive science practitioners in other domains. Thus, identifying a clear gap in the tool support available for interdisciplinary researchers in this expanding field.

For example, the studies using fMRI to evaluate program comprehension [9, 10] have had to rely on tools such as BrainVoyager<sup>1</sup> and SPM<sup>2</sup> which are helpful for the analysis of brain imaging data, but do not integrate other software engineering artifacts, which is vital for modeling cognitive process during tasks such as program comprehension. Recent studies we have conducted use a brain

42

Roy, Devjeet, Sarah Fakhoury, and Venera Arnaoudova. "VITALSE: visualizing eye tracking and biometric data." *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*. 2020.

# VITALSE 1.0

- Interactive visualization of multi-modal biometric data over source code
- Can visualize generic biometric data annotated with eyetracking data
- Useful for exploratory data analysis, qualitative analysis and post experiment interviews



# gazel

- ~2010-2015 Eye tracking data is mapped to **static images or videos** of code
- 2015 ITrace [1] introduces way to track gazes on large source code snippets inside of an IDE, automatic mapping to code elements

**Addresses major limitation:**  
cannot run or edit code,  
limiting the kind of program  
comprehension studies  
researchers can conduct.

```
1 import React, { Component, useState, useEffect, useCallback } from "react";
2
3 class Draggable extends Component {
4   constructor(props) {
5     super(props);
6
7     this.state = {
8       dragEnabled: false,
9       dragStart: null
10    };
11
12    this.onMouseDown = this.onMouseDown.bind(this);
13    this.onMouseUp = this.onMouseUp.bind(this);
14    this.onMouseMove = this.onMouseMove.bind(this);
15  }
16  onMouseUp(e) {
17    document.removeEventListener("mousemove", this.onMouseMove, true);
18    document.removeEventListener("mouseup", this.onMouseUp, true);
19  }
20  componentWillUnmount() {
21    document.removeEventListener("mousemove", this.onMouseMove, true);
22    document.removeEventListener("mouseup", this.onMouseUp, true);
23  }
24  onMouseMove(e) {
25    const dx = e.clientX - this.state.dragStart.x;
26    const dy = e.clientY - this.state.dragStart.y;
27
28    if (this.props.onDrag) {
29      this.props.onDrag(dx, dy);
```

[1] Shaffer, T.R., Wise, J.L., Walters, B.M., Müller, S.C., Falcone, M. and Sharif, B., 2015, August. itrace: Enabling eye tracking on software artifacts within the ide to support software engineering tasks. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (pp. 954-957).

# Supporting real-world tasks

## Snapshot 1

```
60
61 def start_mock_server(
62     frequency: float,
63     duration: float,
64     data: List[dict],
65     timing_fn: Callable[[], float],
66
67     port: int,
68     wait=False,
69     delay=1,
70     data_path="./",
71 ):
72     interval = 1 / frequency
73
```

Fixation 1 mapped to **[dict]** on line 64

Fixation 2 mapped to **delay** on line 68

Fixation 3 mapped to **frequency** on line 71

## Snapshot 2

```
60
61 def start_mock_server(
62     frequency: float,
63     duration: float,
64     data: List[dict],
65     timing_fn: Callable[[], float],
66     new_arg : float,
67     port: int,
68     wait=False,
69     delay=1,
70     data_path="./",
71 ):
72     interval = 1 / frequency
73
```

Fixation 1 mapped to **[dict]** on line 64

Fixation 2 **incorrectly** mapped to **wait** on line 68

Fixation 3 **incorrectly** mapped to **whitespace** on line 71

# gazel: Supporting Experiments with Code Edits

ICSE '21

## gazel: Supporting Source Code Edits in Eye-Tracking Studies

Sarah Fakhoury, Devjeet Roy, Harry Pines, Tyler Cleveland  
Washington State University, USA  
{first.last}@wsu.edu

Cole S. Peterson  
University of Nebraska-Lincoln, USA  
Cole.Scott.Peterson@huskers.unl.edu

Venera Arnaudova  
Washington State University, USA  
venera.arnaudova@wsu.edu

Bonita Sharif  
University of Nebraska-Lincoln, USA  
bsharif@unl.edu

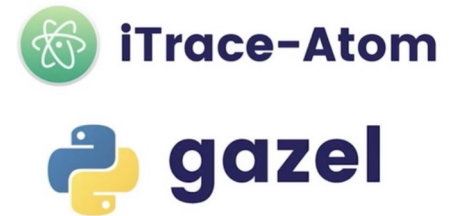
Jonathan I. Maletic  
Kent State University, USA  
jmaletic@kent.edu

**Abstract**—Eye tracking tools are used in software engineering research to study various software development activities. However, a major limitation of these tools is their inability to track gaze data for activities that involve source code editing. We present a novel solution to support eye tracking experiments for tasks involving source code edits as an extension of the *iTrace* [9] community infrastructure. We introduce the *iTrace-Atom* plugin and *gazel* [gə'zel]—a Python data processing pipeline that maps gaze information to changing source code elements and provides researchers with a way to query this dynamic data. *iTrace-Atom* is evaluated via a series of simulations and is over 99% accurate at high eye-tracking speeds of over 1,000Hz. *iTrace* and *gazel* completely revolutionize the way eye tracking

tasks that can be studied using eye trackers (e.g., bug fix or feature addition).

This paper presents a novel solution to address the editing limitation, as an extension to the *iTrace* [9] infrastructure. We propose *iTrace-Atom*, a plugin that tracks gaze and edit information over source code files in the Atom editor, accompanied by *gazel* [gə'zel] (gaze edit evolution) a Python data processing library to analyze the data collected by *iTrace-Atom*.

Researchers can use these tools to track source code elements as they move and change throughout the course of

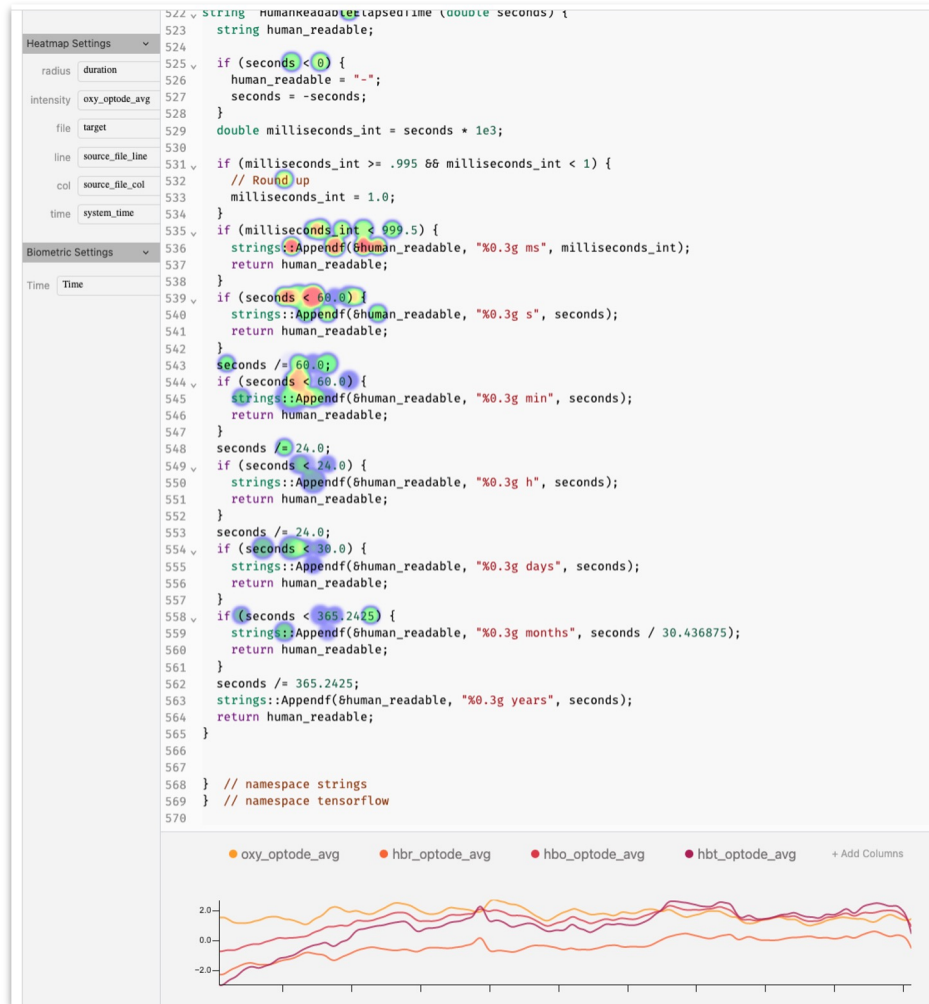


Fakhoury, Sarah, et al. "gazel: Supporting source code edits in eye-tracking studies." *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2021.

# VITALSE 2.0

- Work in progress
- Built to add support for visualization of edits during experiments (evolving source code)

You will be using this in the hands-on today!



# What's next?

- Biometrics provides us with a new lens to view human factors in SE
  - How do we operationalize our findings from biometrics to build tools and languages that support program comprehension?
- Tooling support for biometrics for SE is still in its infancy. We need more community support to build a better ecosystem to support biometric research in SE.
- Advancements in biometrics:
  - The current state of biometrics places many constraints on what we can study. Can we move past the current limitations?



# Hands-on Tutorials

## Getting Started

Download the supporting materials from:

<https://tinyurl.com/SIESTA23-Biometrics>

## Overview of activities

1. Setting Up and Data Preprocessing
2. Introduction to VitalSE 2.0
3. Exercises



WASHINGTON STATE  
UNIVERSITY